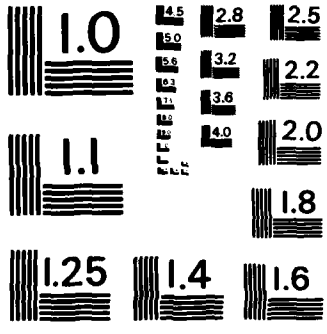
AD-A161 931 FUNCTIONAL TESTING OF LSI/VLSI BASED SYSTEMS WITH MEASURE OF FAULT COVERAGE(U) STATE UNIV OF NEW YORK AT ALBANY RESEARCH FOUNDATION S Y SU 08 FEB 84 1/1

UNCLASSIFIED DAAB07-82-K-J056 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS - 1963 - A

SCIENTIFIC AND TECHNICAL REPORT

~~FIFTH~~ Sixth QUARTERLY STATUS REPORT

Prepared By

Stephen Y.H. Su
Project Director
Dept. of Computer Science
State University of New York
Binghamton, New York 13901
(607)-798-2296 (office)
(607)-798-4802 (secretary)

A. Contractor's name and address:

The Research Foundation of
State University of New York
P.O. Box 9, Albany, New York 12201

B. Contract No. DAAB07-82-K J05-6

C. Date of Report - February 8, 1984

D. Title: The ~~Fourth~~ Sixth Quarterly Status Report for the project "Functional Testing of LSI/VLSI Based Systems with Measure of Fault Coverage"

E. Period Covered: Oct. 28, 1983 to Jan. 27, 1984

F. Description of Progress: See attached

G. Spending: Actual Quarter Cost: $ 7,800.00

Cumulative Cost to date: $ 140,427.15

Functional testing is a feasible solution for LSI/VLSI test generation and design verification. In this paper, we present a systematic way to perform functional testing using an advanced symbolic execution technique. Symbolic execution is a very useful and powerful software engineering technique mainly used in program analysis including test case generation. Often a single symbolic execution of a program may represent a large number of normal test runs. For example, consider the following register transfer level statement:

IF RA(15)=0 then RB $\leftarrow$ RC else RD $\leftarrow$ RC

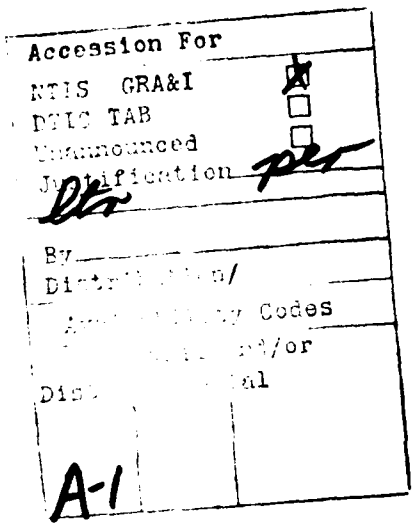
which means: if bit 15 of register A is equal to binary 0, then transfer the content of register C to register B, otherwise do the same action from C to D. When we symbolically execute this statement, we may only specify the value of bit RA(15) (leaving other bits as "don't cares") to activate one of the two branches. This actually means that all input cases with bit RA(15) set to a specified value (either 0 or 1) and other bits, RA (0) to RA (14), unspecified are covered in one symbolic run of the statement. Note that if an actual value of RA is to be used, then there are potentially 2^{15} choices to make!

Most digital systems including complex VLSI devices can be functionally described by the standard Register Transfer Language (RTL). By applying symbolic execution to the RTL description of digital systems, one can systematically derive efficient input test data with fault coverage dependent on the precision of the functional fault model used. The application of symbolic execution in a RTL program is easier than its application in higher level programming languages, since many complicated situations specific in high level languages will never occur in a RTL program. This is because the syntax structure of a RTL statement is usually much simpler than that of a high level language statement.

This technique can be applied in test generation and/or design verification of general digital systems both when the implementation is known and unknown. In the latter case, a RTL program for describing the behavior of a digital system can be derived based on user accessible information such as data book and application notes. Certain inevitable assumptions (and hence functional redundancies) such as the internal logic paths connections, functional separation of internal parallism, or functional derivation of control circuit timing, may be made in this case.

The basic requirement for performing symbolic execution is the RTL description of a digital system under test and a set of functional faults derived from RTL statements. The preprocessing of a RTL program is first performed to partition the RTL program into basic modules, to set up test order among modules, and to collect other control information for subsequent symbolic execution. Each RTL module is tested in the sequence decided by the test order obtained in preprocessing. Each fault-free functional module is symbolically executed once to set up its symbolic execution tree. The input test pattern for a set of testable faults (eg., register content stuck-at-fault) are derived first. Then each remaining functional fault derived from RTL fault model is enumerated and injected into the good machine. Symbolic execution is then performed to produce the symbolic constraints for the faulty machine. An inequality solver is then used to derive precisely the set of test data for the specified fault by performing an "exclusive OR" operation on the constraints for the fault-free and fault-injected execution. More types of practical faults can be considered and included in the RTL fault model.

Since each symbolic execution run always generates all test patterns for a given fault, near-optimal test set may be obtained during the iterative symbolic executions. The symbolic execution of a RTL description of a digital system may also point out design errors in the RTL level. For example, logic designers may adjust their design if certain register transfer paths will never be activated by all symbolic inputs or the symbolic execution produces unexpected results with respect to certain inputs.



CRITICAL PATH TEST GENERATION AT REGISTER TRANSFER LANGUAGE LEVEL

Test generation algorithms at the gate level are not practical for VLSI (very large scale integration) circuits. Design for testability will not help the testing of the existing off-the-shelf digital devices. Therefore, recent effort has essentially been spending devoted in the test generation at higher levels. These functional testing techniques not only reduce the test generation complexity, but also allow us to test VLSI without the need of knowing the circuit implementation.

One kind of path sensitizing techniques is called critical path test generation which were first introduced in the LASAR system (1) and later developed at the gate level (2-5). To increase the efficiency, this method attempts to find a test pattern for detecting as many faults as possible instead of just a given fault. Furthermore, a test pattern and the list of faults detected by the test can be generated simultaneously. Therefore, fault simulation will not be required (2). Recently, the critical path test generation approach has been extended to high level systems (6, 7).

We propose a critical path test generation method for systems based on the Register Transfer Language (RTL) descriptions. A fault model at the RTL level is defined. There are three types of faults: data fault, control fault and operation fault. The critical path approach attempts to use critical cubes of elements to propagate the criticality of lines from prime outputs to prime inputs (critical tracing). We extend the concept of critical cube from gate level to the RTL level, and present three methods to construct critical cubes: truth table method, Boolean difference method and Ad-hoc method.

The critical path method mainly need two procedures: critical tracing and implication. We will present some rules for these procedures. The key idea of the critical tracing is to attempt single path sensitizing. We consider the general use of multiple path sensitizing with reconvergent fan-outs.

We will also discuss how to find faults detected by a generated test. Finally, we will present a critical path test generation algorithm and give an example.

References

(1) J.J. Thomas, "Automated Diagnostic Test Programs for Digital Networks", Computer Design, Aug. 1971, pp. 63-67.

(2) D.T. Wang, "An Algorithm for the Generation of Test Sets for Combinational Logic Networks", IEEE Trans. on Computers, July 1975, pp. 742-746.

(3) D.T. Wang, "Properties of Faults and Criticalities of Values under Tests for Combinational Networks", IEEE Trans. on Computers, July 1975, pp. 746-750.

(4) M.A. Breuer and A.D. Friedman, Diagnosis and Reliable Design for Digital Systems, Computer Science Press, 1976.

(5) M. Abramovici, P.R. Menon and D.T. Miller, "Critical Path Tracing - An Alternative to Fault Simulation", 20th Design Automation Conference, June 1983, pp. 214-220.

(6) Y. Levendel and P.R. Menon, "The *-algorithm: Critical Traces for Functions and CHDL Constructs", 13th International Symposium on Fault-Tolerant Computing, June 1983, pp. 90-97.

(7) M. Kawai, et.al., "A High Level Test Pattern Generation Algorithm", 1983 International Test Conference, Oct. 1983, pp. 346-352.

END

FILMED

1-86

DTIC